Internet Relay Chat Class Project
draft-irc-pdx-jtn4-cs494-01.odt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on Fail 1, 0000.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class at Portland State University.

Table of Contents

1. Introduction

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This system employs a central server which 'relays' messages that are sent to it to other connected users.

Users can join rooms, which are groups of users that are subscribed to the same message stream. Any message sent to that room is forwarded to all users currently joined to that room.

Users can also send private messages directly to other users, as well as transfer files directly to other users that request them.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 17734. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests to the server over this open channel, and the server can reply via the same. This messaging protocol is inherently asynchronous - the client is free to send messages to the server at any time, and the server may asynchronously send messages back to the client.

As is described in [5.2], both the server and client may terminate the connection at any time for any reason. They MAY choose to send an error message to the other party informing them of the reason for connection termination.

The server MAY choose to allow only a finite number of users and rooms, depending on the implementation and resources of the host system. Error codes are available to notify connecting clients that there is currently a high volume of users or groups accessing the server.

4. Name Semantics

Identifying both users and rooms involves sending and receiving names. Name rules are identical for users and rooms, and MUST be validated as follows:

- Field size for transmissions is always exactly 64 bytes.
- Must consist entirely of UTF-8 encoded Unicode points above 0x0020, excluding the directional embedding codes: 0x202A-0x202E, 0x2066-0x2069, 0x200E, 0x200F and 0x061C.
- Must be at least one Unicode point, and at most 32 Unicode points.
- If the UTF-8 encoding of the name is less than 64 bytes, then the first byte following the name MUST be a NULL byte (0x00). Remaining bytes MAY also be NULL.
- If any of these rules are broken, the receiver MUST terminate the connection and MAY provide the IRC_ERR_ILLEGAL_NAME error.

- Before using this field, recipients SHOULD append a NULL terminator to this array to reduce the likelihood of a buffer overflow attack.

5. Message Infrastructure

5.1. Generic Message Format

```
struct irc_header {
     kind: u8;
     length: u32;
}

struct irc_packet_generic {
     header: struct irc_header;
     payload: [u8; header.length];
}
```

5.1.1. Field definitions:

- header.kind - specifies what kind of message is contained within the payload.
-
- Header.length - specifies how many bytes of payload follow the header in this message (exclusive of header size).
-
- Both client and server MUST validate that the length is valid for the kind of message provided. If not, the entity that detects the error MUST terminate the connection, and MAY provide the error code IRC_ERR_ILLEGAL_LENGTH to the opposite party (see error messages section).
-
- Payload - variable length payload. Not used by some messages.

5.1.2. Message Kinds

```
IRC_KIND_ERR                = 0x01
IRC_KIND_NEW_CLIENT         = 0x02
IRC_KIND_HEARTBEAT          = 0x03
IRC_KIND_ENTER_ROOM         = 0x04
IRC_KIND_LEAVE_ROOM         = 0x05
IRC_KIND_LIST_ROOMS         = 0x06
IRC_KIND_ROOM_LISTING       = 0x07
IRC_KIND_USER_LISTING       = 0x08
IRC_KIND_QUERY_USER         = 0x09
IRC_KIND_SEND_MESSAGE       = 0x0A
IRC_KIND_BROADCAST_MESSAGE  = 0x0B
IRC_KIND_POST_MESSAGE       = 0x0C
```

```
IRC_KIND_DIRECT_MESSAGE     = 0x0D
IRC_KIND_OFFER_FILE         = 0x0E
IRC_KIND_ACCEPT_FILE        = 0x0F
IRC_KIND_REJECT_FILE        = 0x10
IRC_KIND_FILE_TRANSFER      = 0x11
IRC_KIND_CLIENT_DEPARTS     = 0x12
IRC_KIND_SERVER_DEPARTS     = 0x13
```

5.2. Error Messages

```
struct irc_packet_error {
     header: irc_header =
          { .kind = IRC_KIND_ERR, .length = 1 };
     error_code: u8;
}
```

5.2.1. Usage

MAY be sent by either the client or the server prior to closing the TCP connection to inform the other party why the connection is being closed. If either client or server receives this message, that entity SHOULD consider the connection as terminated.

5.2.2. Field definitions:

- error_code - specifies the type of error that occurred

5.2.3. Error Codes

```
IRC_ERR_UNKNOWN          = 0x01
IRC_ERR_ILLEGAL_KIND     = 0x02
IRC_ERR_ILLEGAL_LENGTH   = 0x03
IRC_ERR_NAME_IN_USE      = 0x04
IRC_ERR_ILLEGAL_NAME     = 0x05
IRC_ERR_ILLEGAL_MESSAGE  = 0x06
IRC_ERR_ILLEGAL_TRANSFER = 0x07
IRC_ERR_TOO_MANY_USERS   = 0x08
IRC_ERR_TOO_MANY_ROOMS   = 0x09
```

5.3. Heartbeat Messages

```
struct irc_packet_heartbeat {
     header: irc_header =
          { .kind = IRC_KIND_HEARTBEAT, .length = 0 };
}
```

5.3.1. Usage

Provides an application-layer notification of a disconnected peer.

MUST be sent at least once every 5 seconds by both client and server to notify the other party that they are still connected. Both client and server SHOULD watch for these heartbeat messages and consider the other party disconnected if more than some set period of time has elapsed. If such a timeout is used, the time MUST be no less than 20 seconds.

5.4. Graceful disconnects

```
struct_irc_packet_depart {
     header: irc_header =
          { .kind = IRC_KIND_CLIENT_DEPARTS
               <OR> IRC_KIND_SERVER_DEPARTS, length = LENGTH }
     farewell: [u8; LENGTH];
}
```

5.4.1 Usage

Sent by either a client or the server which intends to close their connection.

A server that is shutting down MAY choose to send IRC_KIND_SERVER_DEPARTS to all connected clients. A client that receives this message should consider the connection terminated and MAY display the farewell message to the user. The client MAY choose to resend a connect message to the server.

A client that is disconnecting MAY send IRC_KIND_CLIENT_DEPARTS. A server that receives this message should consider the connection with the sending client terminated and MAY forward the farewell message to rooms that the client had joined.

5.4.2 Field Definitions

- farewell - A final goodbye message sent by the server or client that intends to close the connection..
  - Must consist entirely of UTF-8 encoded Unicode points above 0x0019, excluding the directional embedding codes: 0x202A-0x202E, 0x2066-0x2069, 0x200E, 0x200F and 0x061C.
  - Must be NULL terminated.
  - MUST not contain extra NULL terminators within the payload, they may only be at the very end.

- MUST be less than 12000 bytes long.

6. Client Messages

6.1. First message sent to the server

```
struct irc_packet_new_client {
     header: irc_header =
          { .kind = IRC_KIND_NEW_CLIENT, .length = 64 };
     chat_name: [u8; 64];
}
```

6.1.1. Usage

Before subsequent messages can be sent, a connecting client MUST provide a chat name and identify which version of the protocol they are using.

The server MUST associate the client's chat_name with the socket connection of the user. This message SHOULD be sent only once; if the server receives the message more than once, the server MAY ether ignore the message or terminate the client's connection.

6.1.2. Field Definitions

- chat_name specifies the name that the connecting client wishes to use. It MUST follow name semantics.

- chat_name must not be the same name provided by any other currently connected client. If the name already exists, the server MUST return the error IRC_ERR_NAME_IN_USE and close the connection. The client can then attempt to reconnect with a different name.

6.2. Listing Rooms

```
struct irc_packet_list_rooms {
     header: irc_header =
          { .kind = IRC_KIND_LIST_ROOMS, .length = 0 };
}
```

6.2.1. Usage

Sent by the client to request a list of all the rooms currently occupied by at least one other client.

6.2.2. Response

Server MUST return an irc_packet_list_rooms_response with kind IRC_KIND_LIST_ROOMS_RESPONSE with a list of the names of all currently occupied rooms.

6.3. Joining and Creating rooms

```
struct irc_packet_enter_room {
     header: irc_header =
          { .kind = IRC_KIND_ENTER_ROOM, .length = 64 };
     room_name: [u8; 64];
}
```

6.3.1. Usage

Sent by the client to join a chat room. If no room by that name exists, one is created for the client to join.

Upon joining a room, the server MUST send an IRC_KIND_USER_LISTING message to all users currently in that room to alert them that the user list has changed. The identifier MUST be set to the name of the room, and the user_names list MUST include a list of all of the users in that room.

Every time the room's user list changes the server MUST send a new IRC_KIND_USER_LISTING message to all users in the room informing them of the new room membership.

6.3.2. Field Definitions

- room_name - Name of the room to enter or create. MUST follow name semantics.

6.4. Leaving a Room

```
struct irc_packet_leave_room {
     header: irc_header =
          { .kind = IRC_KIND_LEAVE_ROOM, .length = 64 };
     room_name: [u8; 64];
}
```

6.4.1. Usage

Sent by the client to leave a chat room.

Upon receiving this message the server MUST remove the client from the specified room and MUST send an IRC_KIND_USER_LISTING message to all users currently in that room to alert them that the user list has

changed. The identifier MUST be set to the name of the room, and the user_names list MUST include a list of all of the users in the room.

The server SHOULD ignore leave requests when the client is not currently a member of the specified room.

6.4.2. Field Definitions

- room_name - Name of the room to enter or create. MUST follow name semantics.

6.5 Sending Messages

```
struct irc_packet_send_message {
     header: irc_header =
          { .kind = IRC_KIND_SEND_MESSAGE
               <OR> IRC_KIND_BROADCAST_MESSAGE
               <OR> IRC_KIND_DIRECT_MESSAGE, .length = LENGTH };
     target_name: [u8; 64];
     message: [u8; LENGTH-64];
}
```

6.5.1. Usage

Sent by a client to send a text message to a room, all rooms the user is in, or another user.

If the kind is IRC_KIND_SEND_MESSAGE then the target is a room and, after validating this message, the server MUST send an IRC_KIND_POST_MESSAGE to all users in the specified room with the target_name parameter set to the room name, and the sending_user parameter set to the name of the user who sent the message. The server MAY choose to forward messages to rooms that the client is not a member of.

If the kind is IRC_KIND_BROADCAST_MESSAGE then the target is all rooms the sending user is a member of and, after validating this message, the server MUST send an IRC_KIND_POST_MESSAGE to all users in the affected rooms with the target_name parameter set to the room name, and the sending_user parameter set to the name of the user who sent the message. The target_name parameter given by the sending user is unused. If a recipient user has more than one room in common with the sending user they will be sent multiple IRC_KIND_POST_MESSAGE, one for each room in common.

If the kind is IRC_KIND_DIRECT_MESSAGE, the the target is another user. If the message passes validation, and if the intended recipient is online, the server MUST forward the message by sending an

IRC_KIND_DIRECT_MESSAGE to the specified user. The target_name field of the forwarded message MUST be set to the name of the user who originated the direct message. If the recipient is not online, the server MUST instead send an IRC_KIND_QUERY_USER message to the originating user indicating that the recipient is offline.

6.5.2 Field Definitions

- target_name - Name of the entity to send the message to.
- Message - Message to send to room(s) or individual users.
  - Must consist entirely of UTF-8 encoded Unicode points. Including horizontal tab: 0x0009, and points above 0x0019, excluding the directional embedding codes: 0x202A-0x202E, 0x2066-0x2069, 0x200E, 0x200F and 0x061C.
  - Must be NULL terminated.
  - MUST not contain extra NULL terminators within the payload, they may only be at the very end.
  - MUST be less than 12000 bytes long.
  - MUST contain at least one Unicode point in addition to the null terminator.
  - If any of these rules are broken, the receiver MUST terminate the connection and MAY provide the IRC_ERR_ILLEGAL_MESSAGE error.

6.6. Checking if a user is online

```
struct irc_packet_query_user {
     header: irc_header =
          { .kind = IRC_KIND_QUERY_USER, .length = 65 };
     user_name: [u8; 64];
     status: [u8];
}
```

6.6.1. Usage

Sent between clients and sever to validate whether a particular user is connected to the server. This may be used to manage a friend's list or facilitate direct messaging between users who do not have a room in common.

6.6.2. Response

The server MUST send an IRC_KIND_QUERY_USER message response to the requesting user. The server will indicate the user's online/offline state in the status field.

6.6.3. Field Definitions

- user_name - Name of the user whose online status is being queried. MUST follow name semantics.
- Status - Indicates whether the specified user is currently connected to the server. Servers MUST set this to 0 if the user is disconnected from the server and MUST set this to 1 if the user is connected to the server. Clients SHOULD set this to 2 when requesting user status from a server. Servers MUST accept any value in this field from clients.

7. File Transfers

7.1 Negotiating File Transfers

```
struct irc_packet_transfer_negotiation {
     header: irc_header =
          { .kind = IRC_KIND_OFFER_FILE
               <OR> IRC_KIND_ACCEPT_FILE
               <OR> IRC_KIND_REJECT_FILE, .length = LENGTH };
     target_name: [u8; 64];
     source_name: [u8; 64];
     transfer_id: u16;
     file_size: u32;
     file_name: [u8; LENGTH-134];
}
```

7.1.1. Usage

Sent between clients and the server to negotiate the start of a file transfer. A client first sends IRC_KIND_OFFER_FILE with the name of a file, its size in bytes, the name of a client to whom they wish to send the file, and their own name. transfer_id is unused when sent from a client to the server and SHOULD be zero.

If the receiving user is not online the server MUST reply to the offering client with IRC_KIND_REJECT_FILE with the same values for target_client and file_name and a transaction_id of zero. If the source_name does not match the name of the offering client than the server MUST send an IRC_ERR_ILLEGAL_NAME error to the offering client and close their connection.

If the receiving user is online the server MUST forward the IRC_KIND_OFFER_FILE message to them with the transfer_id changed to a number selected by the server. The value chosen for transfer_id MUST be unique across all ongoing file transfers through the server. The server MUST keep a record of the sending client, receiving client, and transfer_id for the offered file transfer until it is completed, fails to be sent due to either client disconnecting, or is rejected.

After receiving an IRC_KIND_OFFER_FILE message, the receiving client MUST reply to the server with either IRC_KIND_ACCEPT_FILE or IRC_KIND_REJECT_FILE. The field values in the response MUST be the same as those given in the offer message. If the receiving client accepts the file they MUST retain the file name, size, and transfer ID for later use during the transfer process.

The server MUST, after validation, forward any IRC_KIND_ACCEPT_FILE or IRC_KIND_REJECT_FILE message it receives to the offering client. If the source_name, target_name, or transfer_id do not match a file transfer recorded by the server than the server must reply to the receiving client with IRC_ERR_ILLEGAL_NAME and close their connection AND MUST reply to the offering client with a corresponding IRC_KIND_REJECT_FILE message with a transfer_id of zero.

The offering client MUST record the transfer_id and begin sending the offered file if it receives a matching IRC_KIND_ACCEPT_FILE response to its file offer. It MAY resend the IRC_KIND_OFFER_FILE after receiving a matching IRC_KIND_REJECT_FILE response with a non-zero transfer_id. IF the reject message's transfer_id is zero then the intended recipient is not online and the offering client SHOULD not resend the IRC_KIND_OFFER_FILE message.

If a client involved in a file transfer departs gracefully, closes their connection to the server, fails to send heartbeat messages, or is disconnected to due any error the server MUST inform the other client involved in the file transfer with an IRC_KIND_REJECT_FILE message. A client receiving this message after a file transfer started MUST consider the file transfer over and MAY discard the transferred portion of the file.

7.1.2. Field Definitions

- target_name - Name of the user who will be receiving the file.
- source_name - Name of the user who will send the file.
- transfer_id - Numeric ID for the file transfer, chosen to be unique by the server upon receiving an initial IRC_KIND_OFFER_FILE message.

- file_size - Size in bytes of the file to be transferred, maximum is 2^32 - 1 = 4 Gigabytes.

- file_name - Name of the file to be transferred. MUST follow the name semantics with the following modifications:

  - MAY include space characters, Unicode 0x020.

  - MAY be larger than 64 bytes, MUST be smaller than 1024 bytes.

  - MUST not start or end with a space.

  - MUST not contain file system reserved characters ':' 0x003A or '/' 0x002F.

7.2. Performing File Transfers

```
struct irc_packet_file_transfer {
     header: irc_header =
          { .kind = IRC_KIND_FILE_TRANSFER, .length = LENGTH };
     transfer_id: u16;
     finished: u8;
     data: [u8; LENGTH-3];
}
```

7.2.1. Usage

Sent by the client to the server when transferring a file to another user. The server MUST validate that the transfer_id matches an in progress file transfer originating with the sender. If the packet passes validation the server MUST forward a copy of the packet to the receiving user associated with the transfer_id.

If the transfer_id does not match a valid in progress file transfer from the sending client to the receiving client than the server or receiving client MUST reply to the packet sender with an IRC_ERROR_ILLEGAL_TRANSFER and close the connection.

The finished flag is used to signal that the file transfer is complete and the transfer details may be purged by the server.

7.2.2. Field Definitions

- transfer_id - Unique identifier for the file transfer selected by the server during file transfer negotiation.

- finished - Byte flag indicating whether this packet is the final packet to be sent in the file transfer. MUST be 0x00 if more

packets are to be sent. MUST be 0x01 if the final byte of the file transfer is contained in the packet.

- data - Bytes from the file being transferred. MUST be at least 1 byte and MUST be no more than 4096 bytes. Clients SHOULD send the maximum number of bytes possible in each packet until the entire file has been sent.

8. Server Messages

8.1 Listing Responses

```
struct irc_packet_listing_response {
     header: irc_header =
          { .kind = IRC_KIND_ROOM_LISTING
               <OR> IRC_KIND_USER_LISTING };
     identifier: [u8; 64];
     name_list[[u8: 64]; LENGTH/64 - 1];
```

8.1.1. Usage

Generic listing response message sent by the server to inform a client of a list. Used for both listing rooms and listing users in a room.

8.1.2. Field Definitions

- identifier - Used only for IRC_KIND_USER_LISTING, contains the name of the room to which the users belong. MUST follow name semantics.

- name_list - Array of names for users or rooms, MUST follow name semantics.

8.2 Forwarding Room Messages to Clients

```
struct irc_packet_post_message {
     header: irc_header =
          { .kind = IRC_KIND_POST_MESSAGE, .length = LENGTH };
     target_name: [u8; 64];
     source_name: [u8; 64];
     message: [u8; LENGTH - 128];
}
```

8.2.1. Usage

Sent by the server to inform clients of a message posted to a room the user is a member of. Server MUST set the target_name field to the name

of the room to which the message belongs. Clients SHOULD ignore this message if the target_name does not match one of the rooms the client believes it has entered.

8.2.2. Field Definitions

- target_name - Name of the room that the messages was sent to. Must follow name semantics.
- sending_user - Name of the user who sent the message. Must follow name semantics.
- message - Message posted to the room.
  - Must consist entirely of UTF-8 encoded Unicode points above 0x0019, excluding the directional embedding codes: 0x202A-0x202E, 0x2066-0x2069, 0x200E, 0x200F and 0x061C.
  - Must be NULL terminated.
  - MUST not contain extra NULL terminators within the payload, they may only be at the very end.
  - MUST be less than 12000 bytes long.
  - MUST contain at least one Unicode point in addition to the null terminator.

8.3 Forwarding Direct Messages to Clients

```
struct irc_packet_forward_direct_message {
     header: irc_header =
          { .kind = IRC_KIND_DIRECT_MESSAGE, .length = LENGTH };
     source_name: [u8; 64];
     message: [u8; LENGTH-64];
}
```

8.3.1. Usage

Sent by the server to inform clients of a message sent directly to them by another user. Server MUST set the source_name field to the name of the sending user.

8.3.2. Field Definitions

- sending_user - Name of the user who sent the message. Must follow name semantics.

- message - Message sent to directly to the user.
  - Must consist entirely of UTF-8 encoded Unicode points above 0x0019, excluding the directional embedding codes: 0x202A-0x202E, 0x2066-0x2069, 0x200E, 0x200F and 0x061C.
  - Must be NULL terminated.
  - MUST not contain extra NULL terminators within the payload, they may only be at the very end.
  - MUST be less than 12000 bytes long.
  - MUST contain at least one Unicode point in addition to the null terminator.

8.4 Responding to user queries

```
struct irc_packet_query_user {
     header: irc_header =
          { .kind = IRC_KIND_QUERY_USER, .length = 65 };
     user_name: [u8; 64];
     status: [u8];
}
```

8.4.1. Usage

Response from the server to a client indicating whether a user is online.

8.4.2. Field Definitions

- user_name - Name of the user whose online status is being queried. MUST follow name semantics.
- Status - Indicates whether the specified user is currently connected to the server. Servers MUST set this to 0 if the user is disconnected from the server and MUST set this to 1 if the user is connected to the server. Servers MUST accept any value in this field from clients.

9. Errors, crashes, and unexpected disconnections

Both server and client MUST detect when they have communication with the other party. This occurs when the socket connection linking them is terminated (by network timeout, server/client closing the connection abruptly by intent or crash) or has gone idle (no heartbeat messages received from the other party for longer than the permitted timeout).

If the server detects that communication with a client has been lost the server MUST remove the client from all rooms to which they are joined, MUST send IRC_PACKET_REJECT_FILE messages to other clients that are currently transferring files with the departed client, and MUST remove the client's file transfers from the server's record of in progress file transfers.

If a client detects that communication with the server has been lost, it MUST consider itself disconnected, MUST consider file transfers as failed, and MAY choose to reconnect or shut down.

As stated previously, it is optional for one party to notify the other in the event of an error or immediately prior to closing the connection intentionally.

10. "Extra" Features Supported

Note that private (direct) messaging and file transfers are supported in addition to meeting the other remaining project criteria.

11. Conclusion and Future Work

This specification provides a generic message passing framework for multiple clients to communicate or share files with each other via a central forwarding server.

Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the text-passing and binary file transfer systems described here. For example, secure connections using cryptographic transport protocols such as Transport Layer Security (TLS) or file servers with file discovery, upload and download support.

12. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server sees all messages and files that are sent through the use of this service. Direct messaging and file transfers may be easily intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol.

13. IANA Considerations

None

14.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

15. Acknowledgments

This document was prepared by using the CS594-SampleRFC.pdf file provided in course materials as a template.

Authors' Address

James Nichols
Portland State University

Email: jtn4@pdx.edu